LA-UR--93-302

DE93 007368

TITLE: **STRUCTURED ADAPTIVE MESH REFINEMENT ON THE CONNECTION MACHINE**

AUTHOR(S): Marsha J. Berger and Jeff S. Saltzman

SUBMITTED TO: Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing
March 22-24, 1993
Norfolk, Virginia

Structured Adaptive Mesh Refinement on the Connection Machine

Marsha J. Berger[*] Jeff S. Saltzman[†]

Abstract

Adaptive mesh refinement has proven itself to be a useful tool in a large collection of applications. By refining only a small portion of the computational domain, computational savings of up to a factor of 80 in 3 dimensional calculations have been obtained on serial machines. A natural question is, can this algorithm be used on massively parallel machines and still achieve the same efficiencies?

We have designed a data layout scheme for mapping grid points to processors that preserves locality and minimizes global communication for the CM-200. The effect of the data layout scheme is that at the finest level nearby grid points from adjacent grids in physical space are in adjacent memory locations. Furthermore, coarse grid points are arranged in memory to be near their associated fine grid points. We show applications of the algorithm to inviscid compressible fluid flow in two space dimensions.

1 Overview

Over the last ten years, local adaptive mesh refinement has been successfully applied to simulate compressible flow in two and three space dimensions to study a variety of physical phenomena [1, 4]. These studies were mostly on serial and vector architectures. We are re-designing the local adaptive mesh refinement algorithm (henceforth AMR) for massively parallel computers. Briefly, AMR uses a hierarchy of nested grids, superimposed on a given coarse grid, to achieve sufficient resolution in the solution. Procedures have been developed to automatically estimate the error (i.e. determine where the resolution of the solution is insufficient), generate refined subgrids, and set up the data structures to do the necessary interpolations between the grids. Any hyperbolic conservation law and an explicit finite volume integration scheme can be used in conjunction with AMR.

Several other adaptive mesh schemes have been implemented on massively parallel machines [3, 5], all to our knowledge using unstructured meshes. There are different issues facing an unstructured versus a structured mesh method. For an unstructured adaptive mesh scheme, the key issues are how to partition the nodes in order to load balance the computation, and how to redistribute the nodes when the mesh is adapted. For AMR, partitioning and load balancing are not the main issues. We restrict all refined meshes to be the same size,i.e. the same number of grid points but with smaller mesh spacing. In particular we use a size that fits exactly onto a quadrant of the CM. (It doesn't pay to use less than this size on a SIMD machine). We map the individual grid points of a grid on to a processor rather than the coarser grained approach of mapping grids to processors. Since

[*]Courant Institute, 251 Mercer Street, New York, NY 10012

[†]Los Alamos National Laboratory, MS B265, Los Alamos, NM 87545

each grid is locally rectangular, it can by itself be integrated with high efficiency during the course of the time stepping.

The two key issues for AMR are: (1) how to map the structured grids onto a massively parallel machine in a way that minimizes inter-grid communication and preserves locality, and (2) how to choose the above mentioned fixed grid size so that it is large enough to get good performance yet small enough so that adaptivity still pays. To take care of the first issue we have developed a data layout scheme that keeps grid points on neighboring grids at the finest level in neighboring processors on the CM. Also, the coarse grid points are within a small distance ($\leq$ the refinement ratio between grids levels, typically 2 or 4) from the fine grid point they need to communicate with. We can preserve locality and still minimize global communication. We were less successful with the second issue however, in part due to some deficiencies in the compiler on the Connection Machine. In two space dimensions, we had to use a larger fine grid size (64 by 64) than the 32 by 32 size we would have preferred. However in work in progress in three dimensions we expect these tradeoffs to change.

2 Data Layout Scheme

Our data layout scheme determines how both the finest level grids and the coarser grid levels are mapped to processors. We describe the fine level mapping first. Consider the CM as a mesh machine with as many processors as grid points (using virtual processors if necessary). Each space dimension is mapped independently as a tensor product of the one dimensional mappings. Suppose first there is a single fine grid in a two-level computation. Let the grid size be 32 by 32 (for purposes of exposition), as well as being the size of the processor array. Point $(1,1)$ on the grid is mapped to a chosen processor (i,j) on the machine, and the rest of the grid is then periodically wrapped around (in both dimensions). In other words, when the grid points run out of processors in their direction, they wrap around to the beginning of the processor array and continue. The starting processor location (i,j) is chosen like this: Determine what the coordinates of the point $(1,1)$ on the fine grid would be if the whole domain were completely refined into one large fine grid; call this (k,m). The starting processor for the periodic wrapping is

$$(1) \qquad i = (k-1) \bmod 32 + 1; \quad j = (m-1) \bmod 32 + 1.$$

In this way, adjacent points belonging to neighboring grids are mapped to neighboring processors.

The coarse grid is also periodically shifted in the same manner as the fine grid. In addition, the coarse grid permutes its grid points so that each coarse point is near its refined points. We give a simple example of it here, the complete description is in [2]. If the grid is refined by 4, then four fine grid points are located under each coarse cell. If the coarse grid starts at point $(1,1)$ on processor $(1,1)$, the 32 grid points are laid out (in one dimension) as follows:

```
1  9  17 25 2  10 18 26 3  11 19 27 4  12 20 28 5  13 21 29 6  14 22 30 7  15 23 31 8  16 24 32
```

Compare this to the fine grid starting at $(1,1)$ on processor $(1,1)$:

```
1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
```

Notice for example that point 17 on the fine grid is on the same processor as cell 5 of the coarse grid. In general, point (u,v) on the coarse grid is first mapped to processor

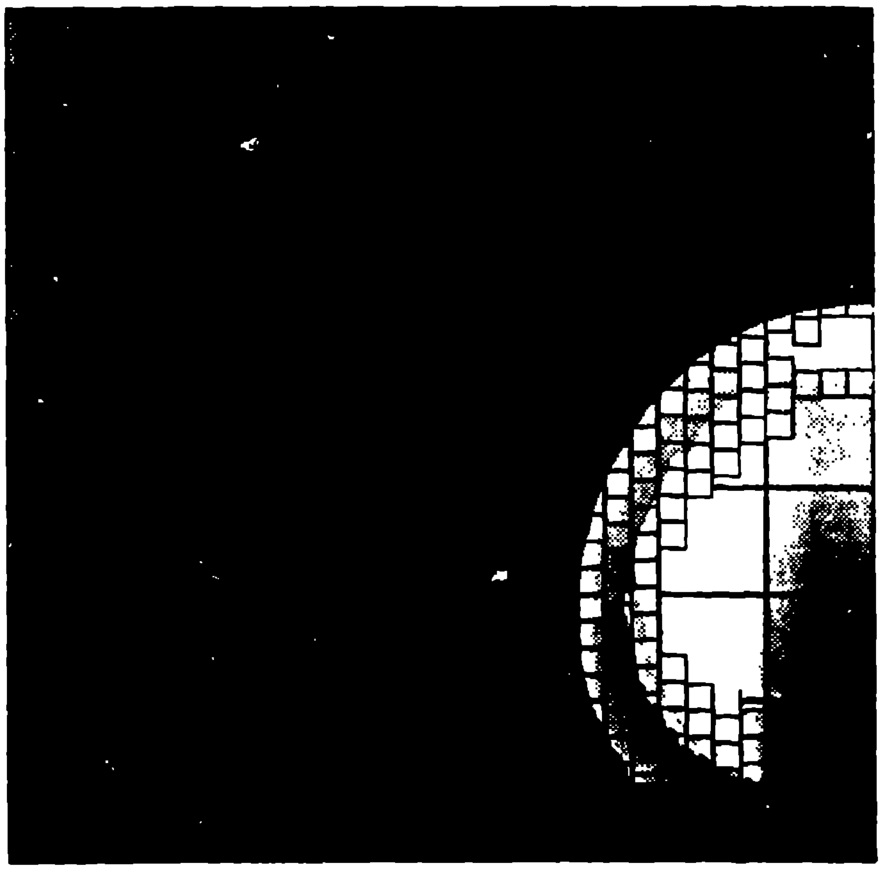

FIG. 1. *The log of the density is plotted using a spectral color raster representation (blue is low density while red is high density). There is a total of 118 tiles at all levels. Density variation is from* 10^{-4} *to* $10^{1.5}$

(u', v') using expression (1), and then is permuted to processor (i, j) with

$$i = (r * u') \bmod 32 + (r * u')/32 + 1; \quad j = (s * v') \bmod 32 + (s * v')/32 + 1.$$

The coarse grid is kept in this shuffled arrangement except when it is playing the role of a finer level grid with respect to an even coarser grid, in which case it is temporarily unshuffled and reshuffled. Note however that the coarse grid permutation depends only on the refinement ratio, and not the actual location of any coarse grid. Therefore, we can use the communication compiler to compute an optimized routing to perform the permutation before the calculation begins. This greatly reduces the permutation communication time; it is only a few percent of the overall runtime.

3 Computational Example

The 2D algorithm is applied to solve an integrated circuit trenching problem using the equations of gas dynamics on the CM 200. These problems arise in developing hybrid integrated circuits. Lasers are used to blast out substrate material so that conductive material can be deposited to connect separate integrated circuit devices. The question of interest is to find out where the debris from the laser blast is deposited.

We model this as thin layer of material heated to a very high temperature in a deep depression (15 microns). The depression is fifteen times the thickness of the heated layer (1 micron). The size of the entire computational region is 112 × 112 microns. The width of the trench is also 15 microns. The right side of the computation region has a symmetry boundary condition (so that the effective width of the trench is 30 microns) . The trench and the remaining computational boundaries use reflecting boundary conditions. (In figure 1 , the black region next to the trench represents part of the substrate, and is not part of the computational domain).

The thin material region at the bottom of the trench is heated to 16,700 degrees Kelvin by the laser while the rest of the problem domain is at room temperature (300 degrees Kelvin). The initial density of the laser heated material is 1.4 grams/cc while the remaining problem domain has a density of 0.00015 grams/cc. Because of the problem parameters, an ideal equation of state can be used, treating the problem as expanding gas into a near vacuum.

Figure 1 shows the solution at a time of 4.0 nanoseconds, after 300 coarse grid time steps. The leading bow shock is followed by a strong shear layer, with some turbulence located at the corner of the trench. The horizontal extent of the shear layer gives an indication of the size of the debris field at late times. At this point, only 10% of the domain is covered by the finest level meshes in this calculation. However, we estimate the overall gain in efficiency is 5. This is due partly to the overhead of AMR (approximately 35% of the CPU time), and the fact that 64 × 64 patches are integrated with less efficiency than say 256 × 256 or larger patches that would be used in uniform mesh calculations (approximately 25% slower). At earlier times there are fewer fine patches in the calculation so the overall saving from using AMR is higher. (Of course this is very problem dependent).

The computations carried out in this example match the qualitative behavior of experiments quite well. Further work is in progress to make the algorithm more efficient and to debug the 3-D version of our work. In the course of developing the parallel algorithm we found opportunites for additional parallelism that we could not exploit because of immature software and the restrictions of the SIMD architecture. For instance, we currently integrate and interpolate one patch at a time even though all patches at a given level can have these operations performed simultaneously. Improvements to compilers on the CM-200, or the message passing capabilities on the CM-5 may lead to even better performance.

References

[1] J. Bell, M. J. Berger, J. Saltzman, and M. Welcome. Three dimensional adaptive mesh refinement for hyperbolic conservation laws. Preprint UCRL-JC-108794, Lawrence Livermore National Laboratory, Livermore, CA, December 1991.

[2] M. J. Berger and J. Saltzman. AMR on the CM-2. Preprint 92.16, RIACS/NASA Ames Research Center, Moffett Field, CA 94035, August 1992.

[3] J. Cerutti and H. Trease. The free-Lagrange method on the Connection Machine. In W. C. H. Trease, J. Fritts, editor, *Proceedings of the Next Free-Lagrange Conference*, Copper Mountain, Colorado, 1991.

[4] E. G. Puckett and J. Saltzman. A 3D adaptive mesh refinement algorithm for multimaterial gas dynamics. *Physica D*, 60:84–93, 1992.

[5] R. Williams. Performance of dynamic load balancing algorithms for unstructured mesh calculations. *Concurrency: Practice and Experience*, 3(5):457–481, October 1991.